The Threnoscope

A Musical Work for Live Coding Performance

Thor Magnusson
Faculty of Arts
University of Brighton
Brighton, United Kingdom
T.Magnusson@brighton.ac.uk

***Abstract*—This paper introduces a new direction in the field of artistic live coding where musical works are presented as pieces in the form of a live coding system. The system itself and the code affordances become equivalent to score system in an open musical work for strong improvisation.**

***Index Terms*—Live coding, music, musical work, interface**

I. Introduction

This year is the tenth anniversary of organized live coding in the arts (see www.toplap.org). Diverse live coding systems, festivals, club nights, conference tracks, journal issues, and research projects have appeared in the last decade that have made this form of practice well known within the fields of art, music and science (in particular computer science). With its foundation in musical performance, live coding has now become common in visual arts, light systems, robotics, dance, poetry, and other art forms that can operate with algorithmic instructions. Although the coding languages differ, interpreted programming languages are typically used, although some have coded in C or Java, and others made use of paper drawings, written or even verbal instructions.

Journal and conference papers on live coding, most often written by the practitioners themselves, have defined the practice [6][28][16], explored it in a computer science context [24][3][17], described particular systems and solutions [25][27][18][10][14], explored live coding as musical scores [3][15], and described it as an embodied musical practice that requires practicing just as acoustic instruments do [26][7][1]. After a decade of fruitful experiments and global dissemination of practices, this paper will identify a new direction in live coding that can currently be detected, namely where systems are designed with increasing constraints, to the degree that they can be seen as musical pieces in themselves.

II. Manifestations but no definitions

Live coding in the field of arts is a strongly heterogeneous practice and thus hard to define. It is evident that live coding is not a specific genre or a practice, but rather a heterogeneous multiplicity of practices that have one thing – the codeness – in common: that algorithmic instructions are written in real-time. Or as McLean defines it, "Rules must be explicit. We may be inventing and changing rules all the time in our heads, but unless those rules are written down and modified while they are being followed by a computer (or other agent), that is not live coding." [16]. Collins states that "[t]he more profound the live coding, the more a performer must confront the running algorithm, and the more significant the intervention in the works, the deeper the coding act." [7]. For Collins, most performances fail to "live up to this promise."

Live coding is therefore about the composition of music (or other art forms, including games) where notation is written in the form of algorithms; about writing step-by-step rules for machines or humans to execute. Finally, it is an empirical fact that most live coding is communicating something in a form other than the code itself. The coding results in music, visuals, dance instructions, games, robotics, etc. Of course, one could imagine a purely conceptual live coding without any other output than the code itself, but I am not aware of such work in the field of live coding, although they do exist in offline coding, for example by Pall Thayer's Microcodes [21].

What about the liveness? Here we enter a more difficult area, as the term might signify two things: that the act is happening in front of live audience (and thus the polite gesture of projecting the screen to that audience), or that the act is simply part of a setup where the coder can change instructions live without having to recompile or restart the program. An archaeology of live coding would show that the former meaning quickly established itself and became the accepted connotation, although live coding does certainly not have to be in front of audience. In fact Fabrice Mogini, an early practitioner of live coding in the field of art, recently described how he uses live coding in his own practice as "bringing together improvisation and composition. The real-time feedback while editing code is useful to prevent compositional systems from getting out of hand and forgetting about perception (eg: serialist techniques)" [19].

Many (McLean, Nilson, etc.) argue that a true liveness also requires that the performer not only manipulates prewritten code in real-time, but actually writes and changes algorithms during the execution of the program. This is arguably a fair condition of live coding, as simply running code and changing variables within a prewritten program in real-time is more comparable to operating buttons, sliders and knobs on a screen interface or a MIDI controller. This stricter definition of live coding is more contended and would result in the fact that some "live coding" performances do not include any live coding at all. Interestingly, the tension between the strong and

weak definition of live coding happens to be strikingly analogous to the dichotomy between composition and performance in written sheet music since the 19th century.

III. Constrained Systems for Live Coding

All live coding systems allow the programmer to write and alter the program in real-time. This is the most fundamental condition of a live coding system, rendering compiled languages, such as C or Java, problematic in this context. Typically, live coders have made use of programming languages such as Perl, Python, Lua, JavaScript, Scheme or SuperCollider some of which were general programming languages but others domain specific (SuperCollider being an example of a dedicated computer music programming language). For reasons of demonstration, novelty and performance, the 'blank slate' has become the norm [22][18] of live coding performances. It means that the performer starts with an empty text document and writes the program "from scratch." This is not an essential requirement, but an understandable practice since 'full slate' coding [4] might obscure whether the recital is actually an algorithm-writing or a parameter-tweaking performance, the former preferred in live coding circles for reasons mentioned above.

The blank slate is always bound to be a relative concept. For example a blank slate in my own ixi lang [14] is impregnated with more musical sounds and patterns than the blank slate of SuperCollider, which in turn is more pregnant than C/C++. This reflects the system's hierarchy of code: ixi lang is written in SuperCollider which is written in C/C++. Live coders have always used their own libraries and convenience classes to make live coding faster and less of an inventing-the-wheel-in-front-of-live-audience process.

Increasingly people package these systems as self standing live coding environments, a unique system derived from that person's coding style. Specific systems such as LOLC [10], Gibber [23], Al-Jazari [12], Scheme Bricks [17], or Texture [18], are all good examples of constrained and limited systems that explore a particular idea, yet providing a wide scope for general musical expression. ixi lang is an example of such a system: it was released in 2009 and has received widespread distribution and use. Users opt to register their emails when downloading the system, allowing me to conduct surveys, studying the use of the language. A survey was conducted with ixi lang users and reported on in the paper "ixi lang: A SuperCollider Parasite" [14]. The survey demonstrated that users enjoy the constraints of the system and find its limitations inspiring (e.g., "a great catalyst for a new project").

Recently, I have begun to explore creating even higher level live coding systems, where the system can hardly be conceived of as a general environment for musical expression, but more comparable to a score or a musical piece. The design of such live coding framework is based on principles that relate to the open work as Eco defined it in the early 1960s [9]. For Eco, the open work can be exemplified as a musical piece that allows for diverse interpretations; an approach found in various late 20th-century works, such as Karl-Heinz Stockhausen's *Plus Minus*, Terry Riley's *In C*, La Monte Young's *Dream House*, or John Zorn's *Cobra*. This new representation of the ontology of the musical work imposes new roles for the interpreter of the piece, a fact picked up by Roland Barthes in 1971, "We know today that post-serial music has radically altered the role of the 'interpreter,' who is called on to be in some sense the co-author of the score, completing it rather than giving it 'expression.'" [2].

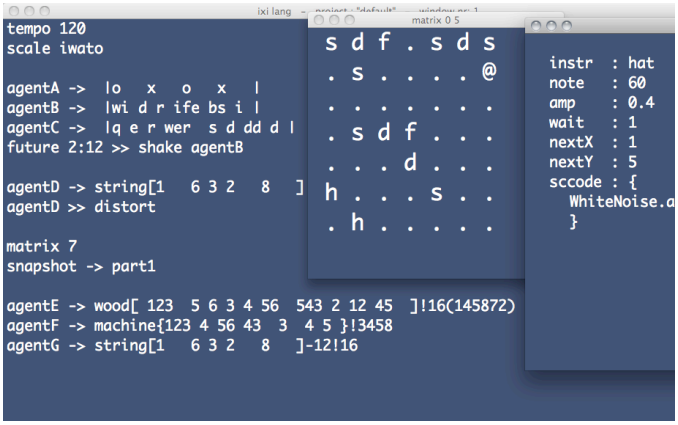

Fig. 1. A screenshot showing the normal ixi lang and the "ixi lang matrix" where agents jump between nodes running SuperCollider code.

When the live coding system in the form of a software package becomes so narrow and focused in its musical scope, it is questionable whether distributing it resembles more the dissemination of a musical score, and if so, whether the user of the system becomes an "interpreter" of the piece. There are many predecessors that have paved the way for this exposition: making art in the form of software is an old and established practice and software artists have published their software for decades now [11]. Instruction pieces also have a strong tradition in the 20th century, Young and Ono being good examples [13]. Recently, Nick Collins published scores for such instruction works, but this time with a strong live coding elements [8]. However, it is uncommon that a live coding system is released in the form of a software; as a musical piece to be interpreted by human performers other than the creator of the piece. This is one of the research aims behind the Threnoscope project presented in the next section. As an open work, in Eco's definition, it allows for wide interpretive scope, like many of the instruction works of the 60s, e.g., Cardew's Scratch Orchestra [5]. However, it is yet to be seen whether indeed users of the system do agree with the definition of the system as a musical piece or whether they see it as a more general live coding platform. The dissemination of this piece will be subject to future research.

IV. The Threnoscope

Issues of time, latency, and sample rate have been of central concern to computer music. All live coding systems contain solutions as to how musical events are scheduled in time and those are often quite original solutions. In ixi lang alphabetical characters are used to represent events (either numbers for notes or letters for sounds) and the spaces between them are non-eventful or silent. The code thus gets a graphical, spatial,

and score-like representation of the music. Since the system updates the written code according the algorithms running, it is concurrently a prescription and description of both the music and the system's state. The ixi lang typically encourages the user to work with beats, polyrhythms, and agents of varied tempi. However, a conscious effort has been made to lessen this event-based focus of ixi lang, for example with the concrete mode and the morphing of samples in the rhythmic mode [14]. Whilst this system has served me well, and I have performed diverse musical sets with it, a fallow musical ground seemed to lie ahead - resulting from a long-lasting interest in drones and microtones - and I wanted to explore it further by designing a specific system addressing certain musical topics.

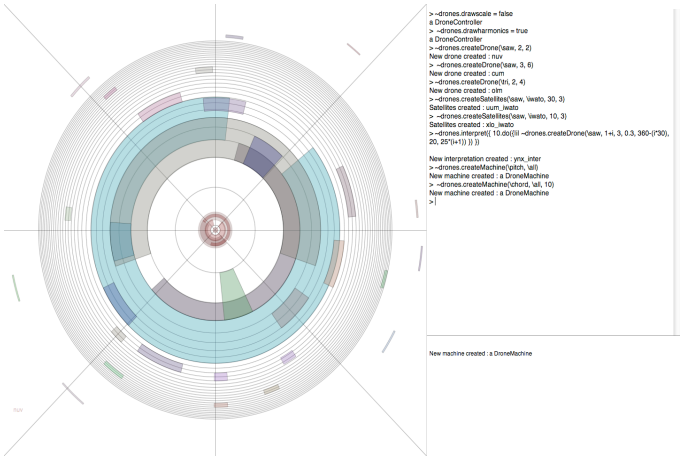

Fig. 2. The Threnoscope live coding system

An extended interest in tuning systems, spectralism, scales, and drone music in general, became the conceptual foundation for the Threnoscope system. The piece is designed for live coding of drones where affordances are provided for manipulations of spectra, wave form, filtering, tuning, and envelopes. Refraining from linear representation of music, the circular interface gives connotations of stasis as the drones slowly circumnavigate the pitch centre. The work focuses on representations of timelessness, the eternal, and emphasizes space rather than time. The drones are harmonic waveforms that can be instantiated anywhere on the circle, but their state can change and pitch, harmonics, resonance, direction and spread, are all properties that can affected by the performer, by pre-composed scores, or by rule-based agents – called drone machines – that appear in the middle of the circle.

The Threnoscope is designed for up to 8 channel speaker systems (but also supports 7.1, 5.1, quadrophonic, stereo, and mono), where virtual output speaker channels are arranged in a circle around a central point. In a typical club setting, a two channel system is used, where a performer sits on stage performing the piece. However, using the eight channels, this circular interface questions the boundaries between a musical piece and an installation: projected on a lateral plane, the surrounding speakers engulf the audience that are able to observe the piece's movement, affected by the behavior of the drone machines. In this manifestation, the piece has no particular beginning or ending, it becomes an attempt to engage with space, happily ignoring its existence in time.

The performance interface is split into three areas. On the left there is the visual score representation with drawn circles of harmonics from the fundamental note that defaults to an A1 (55 Hz). The circles can also display scales of any tuning and scale including custom made ones. Crossing the circles are lines representing the speakers; if a drone intersects a speaker line, it starts sounding out of that speaker. On the right there is the live coding terminal, where the performer can write any SuperCollider code in addition to specific code for the Threnoscope. Below the terminal is the post window, displaying the state of the system and error messages.

The drones themselves can have various waveforms (a saw, triangle, square, noise, sample, etc). They are color coded according to the waveform type. The initial arguments when a drone is created is the wave form, ratio (from fundamental), harmonics, location, size, speed, etc. The drones represent a complex synth instance from a SuperCollider Server synth definition and any of the parameters of that synth can be controlled. The drone can be controlled from code, from the visual interface, and through a network. Since all these parameters can be so easily defined, the drone can range from a static drone that spreads over all the speakers, to a fast moving small drone that triggers quick sounds when crossing speaker lines. Whilst it is theoretically possible to perform the same music with the Threnoscope system and ixi lang, the systems are constructed from a radically different musical concepts, "suggesting" particular ways of thinking and performing.

Since controlling each drone in a live performance can be time consuming and tedious, extra control is given to the system itself. Various automation machines of different functionalities can be initialised that work on the properties of the drones, such as pitch, amplitude, harmonics, etc. The machines appear in the middle of the circle, and they can be live coded themselves, allowing the user to define specific behavior according to needs. The idea of using machines to contribute to the performance is a result of a frustration with the fact that in live coding the coder typically only works on one thing at a time, whereas in musical performance using acoustic instruments, the performer is able to exert multidimensional control through complex bodily gestures.

Further parameters can be exposed to the score interface itself, allowing the live coder to set parameters of the drones through drag-handles, but also draw animation trajectories of those parameters; something that would take considerably more time to code algorithmically.

It is my belief that the performer and the audience gain considerable from the visual representation of the system's state. It is as if the descriptive score confirms what the ears are hearing, but conversely, the ears can tune into specific frequencies or events by cues from the interface. Since the Threnoscope has such compositional constraints in its character I don't hesitate to call it a musical piece, perhaps echoing the way Mumma thinks of himself as a "composer who builds his own instruments, though most of [the] 'instruments' are inseparable from the compositions themselves." [20].

V. Conclusion

This article has described live coding as a multitude of practices. It is a fuzzy concept that does not share any one essential requirement, except perhaps that algorithms are created live. This live coding can equally be aimed at machines or not; written in text or not; with artistic purpose or not; in front of an audience or not; by a programmer or not: it just has to be live algorithms.

Therefore, an attempt to analyze and address the problems of live coding would necessarily apply only for some systems and not others and be relevant or interesting to some coders and not others. In my own live coding practice and in the design of the ixi lang and the Threnoscope, I have tried to address what I consider intriguing topics in live coding system design:

- representation: how is the code visualized?
- speed: can the coder quickly make the desired music?
- audience communication: is the code understandable?
- learning curve: is the system easy to learn?
- control: can processes be easily started and stopped?
- undo: can code be reverted if a mistake was made?
- snapshots: can you go back and forth in time?
- audiovisuality: does the visual element represent the auditory element?
- state update: if an algorithm rewrites the code itself, does the displayed code represent the new state?
- automation: can the system easily run automated processes and how can those be visually represented?
- open theory: can a high level system be flexible and open in terms of tunings, scales and time signatures?
- instrument or a piece: in what sense is the coding system an instrument or a musical piece?

It is clear that the above presentation of live coding systems, and their "problems," might not be conceived as such by other live coders, and if even if they were, the solutions might be completely different from what the two discussed systems present. This is why live coding can be such an interesting field of research and practice: it deals with problems of human-machine interaction; of human and machine languages; and extremely diverse strata of issues in art, performance, and musical composition. A creative solution in a live coding system or a live coding performance is likely to be of an historic interest, due to how young the field is, and how extremely broad and heterogeneous the practices within it are, touching equally upon the arts and the sciences.

References

[1] Aaron, S., A. F. Blackwell, R. Hoadley, and T. Regan. 2011. "A principled approach to developing new languages for live coding." Proceedings of NIME. Oslo: University of Oslo.

[2] Barthes, R. 1978. Image-Music-Text. New York: Hill and Wang. p. 163.

[3] Blackwell, A. and N. Collins. 2005. "The Programming Language as a Musical Instrument." Proceedings of PPIG05 (Psychology of Programming Interest Group).

[4] Burnard, P. 2012. Musical Creativities in Practice. Oxford: Oxford University Press.

[5] Cardew, C. 1972. Scratch Music. London: Latimer New Dimensions Ltd.

[6] Collins, N., A. McLean, J. Rohrhuber, and A. Ward. 2003. "Live coding in laptop performance." Organised Sound, 8(3):321-330.

[7] Collins, N. 2011. "Live Coding of Consequence." Leonardo 44(3):207-211.

[8] Collins, N. 2012. "Six Live Coding Works for Ensemble" Available online at http://www.sussex.ac.uk/Users/nc81/livecodingworksforensemble.html. [Jan 2013].

[9] Eco, U. 1989. The Open Work. Cambridge: Harvard University Press.

[10] Freeman, J., and A. Van Troyer. 2011. "Collaborative Textual Improvisation in a Laptop Ensemble." Computer Music Journal, (35)2:8-21.

[11] Goriunova, O., and A. Shulgin. 2004. read_me: Software Art & Cultures. Aarhus: Aarhus Universitetsforlag.

[12] Griffiths, D. 2007. "Game Pad Live Coding Performance." In J. Birringer, T. Dumke, K. Nicolai. eds. Die Welt als virtuelles Environment, Dresden: TMA Hellerau.

[13] Kotz, L. 2001. "Post-Cagean Aesthetics and the Event Score." October 95. Winter 2001. Pp. 101-140.

[14] Magnusson, T. 2011a. "Ixi lang: a SuperCollider Parasite for Live Coding." Proceedings of ICMC. Huddersfield. UK.

[15] Magnusson, T. 2011b. "Algorithms as Scores: Coding Live Music." Leonardo Music Journal. Leonardo/ISAST.

[16] McLean, A. 2008. "Live coding for free." In Floss+Art. London: Mute Publishing Ltd.

[17] McLean, A., and G. Wiggins. 2010. "Bricolage Programming in the Creative Arts." Proceedings of PPIG 2010 (Psychology of Programming Interest Group).

[18] McLean, A., and G. Wiggins. 2011. "Texture: Visual notation for the live coding of pattern." Proceedings of ICMC.

[19] Mogini, F. 2013. Email on the livecode list 10 jan 2013.

[20] Mumma, G. 1967. Creative aspects of live-performance electronic music technology. Audio Engineering Society Preprint, 33rd Convention. Pp. 348-350

[21] Myers, R. 2009. "Microcodes." Furtherfield, Available online http://www.furtherfield.org/reviews/microcodes. [Jan. 2013].

[22] Nilson, C. 2007. "Live Coding Practice." Proceedings of the NIME Conference. New York.

[23] Roberts, C., and J. Kuchera-Morin. 2012. "Gibber: Live Coding Audio in the Browser." Proceedings of ICMC. Ljubljana.

[24] Rohrhuber, J., A. de Campo, and R. Wieser. 2005. "Algorithms Today: Notes On Language Design for Just In Time Programming." Proceedings of ICMC, San Francisco.

[25] Sorensen, A. 2005. "Impromptu : an interactive programming environment for composition and performance." Proceedings of the ACMC. Queensland University of Technology, Brisbane.

[26] Sorensen, A., and A. Brown. 2007. "aa-cell in practice: An approach to musical live coding." In Proceedings of ICMC.

[27] Wakefield, G., W. Smith, and C. Roberts. 2010. "LuaAV: Extensibility and Heterogeneity for Audiovisual Computing." Proceedings of the Linux Audio Conference (LAC).

[28] Ward, A., J. Rohrhuber, F. Olofsson, A. McLean, D. Griffiths, N. Collins, and A. Alexander. 2004. "Live Algorithm Programming and a Temporary Organisation for its Promotion." In Goriunova, O., and A. Shulgin. Eds. read_me – Software Art and Cultures. Aarhus: Aarhus University Press.